Characterization of HPC Performance and HPC Reference Benchmarks

July 16/17, 2001

Performance Workshop, Oakland

HPC Performance Reference

- To evaluate performance we need a *frame of reference* in the performance space
- This can be established by a set of benchmarks
- All we have right now is peak performance and Linpack

HPC Performance Reference

- Such a reference should be designed to be useful:
  - Across several generations of different HPC architectures
  - For different application domains
  - For a significant period of time

HPC Performance Reference

- Requirements:
  - Architecture independent
  - Scalable code and problem definition
  - Stable performance characteristics
- We need to understand what are the critical aspects of algorithms which determine their performance

Overview

- Characterization of algorithmic features related to performance ("my 1st take")
- Design principles for reference benchmarks

Characterizing Performance

- Characterize performance behavior of applications and algorithms independent from hardware!
- "Time to solution =
  Algorithmic Complexity
  + Data Access Characteristics
  + Structure of Operations"

Algorithmic Complexity

- Theoretical simple:
  - # of Flop or
  - # of Algorithmic Operations
- In Practice we often use hardware counts on specific systems to replace theoretic op-counts
- Why do I mention it?
  - It should not “be part” of a Reference Benchmark

Data Access

- Which features of data-access (DA) influence performance the most?
  - (And by DA I mean local as well as global)
- Locality
  - "Spatial locality of DA"
  - "Temporal locality of DA"
- "Dimensionality of DA"

“Spatial Locality”

- “Can I use multiple contiguous stored data elements in succession?”
  - (Is there a storage scheme which allows this?)
  - Vector-length
  - Data-structure size
  - Message length

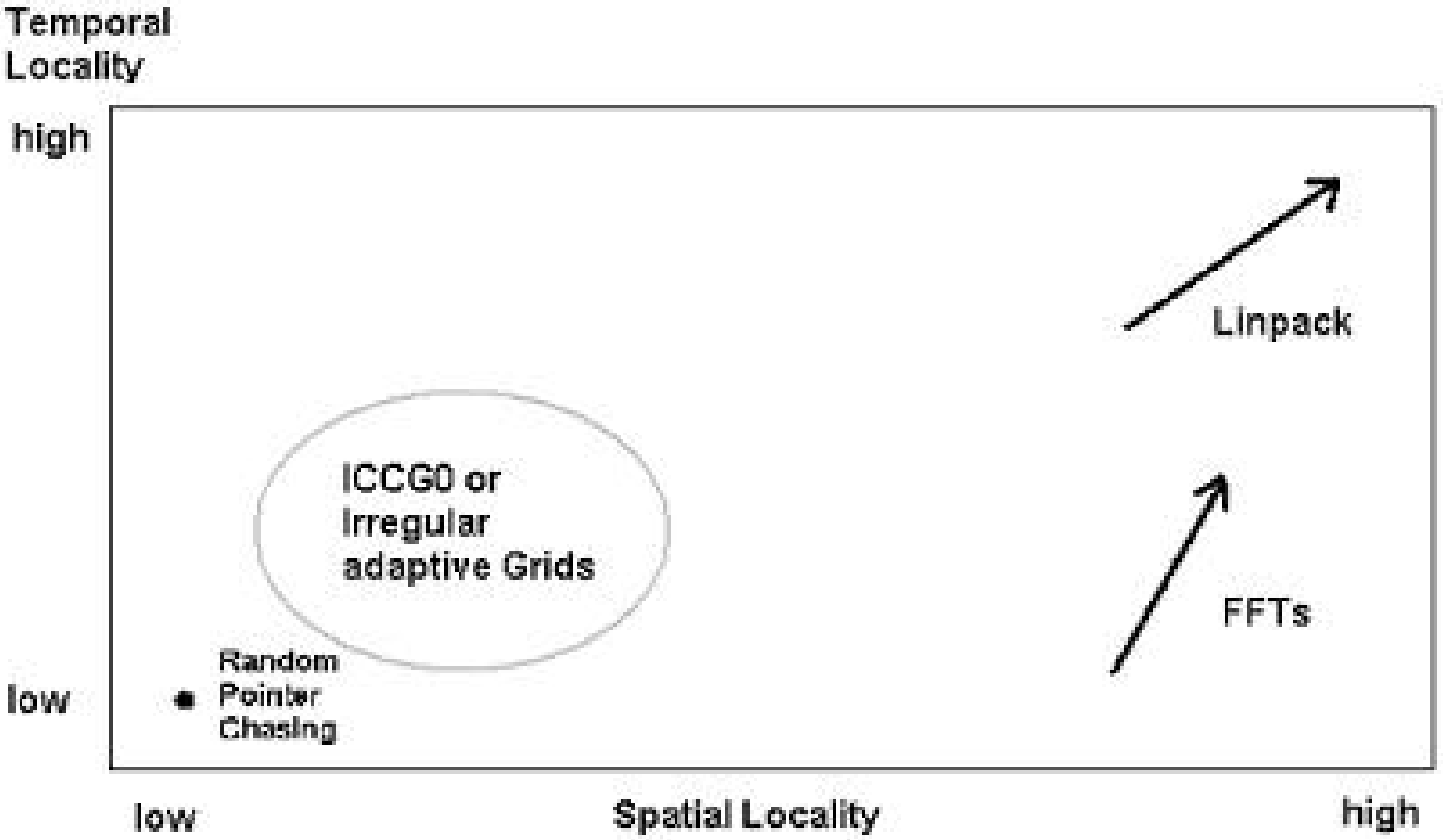
“Spatial Locality”

- Hardware to exploit this:
  - Load/Store (vector) pipelines
  - Cache transfer in “lines”
  - System networks with high bandwidth for long messages

“Temporal Locality”

- “Can I re-use recently accessed data elements?”
  - (Can I re-order my instructions for this?)
- Hardware:
  - Caches
  - (Register)
  - Memory hierarchies in general
  - Distributed memory in particular

(Qualitative) Locality Map



“Data Access Dimensionality”

- “How well can I restructure my data access to increase data locality?”
- Software:
  - “Blocking” to exploit surface to volume effects
  - Reordering of data to assemble long messages
- Essential so we can define a different concept of locality independent of problem size

Structure of Operations

- Operations can be distinguished by influence on control streams:
  - Local Operations:
    - FP, int etc
  - Global Operations:
    - Barrier synchronization
    - Reduction
    - Broadcast

Structure of Operations

- Local Operations
  - "How many (local) Operations can I perform for each data access?"
- Global Operations
  - "How many local Operations can we perform for each global operation?"

Reference Benchmarks

- Key Requirements:
  - Architecture independent
  - Scalable code and problem definition
  - Stable performance characteristics

Architecture Independent

- Pencil and Paper
  - Provide reference implementations
- Complex enough to reflect the influence of all system attributes of interest
- Simple enough to be usable and maintainable

Scalable Definition

- A benchmark useful for the ‘TOP500 class’ of systems for 10 years must bridge a range of **100,000** in system size!
- Benchmarks should utilize resources on a variety of system sizes fully
- Runtimes stable for several generations of systems
  - take algorithmic complexity out

Stable Characteristics

- Performance attributes independent of problem size
- But for this we need to understand what the critical performance attributes are

Design Principles

- Benchmarks for ‘dimensions’
  - Simple to design and understand
  - Eliminates ‘interactions’ between dimensions
  - Opens door for over-optimization
- Simulated application benchmarks
  - Harder to design and understand
  - More realistic performance (hopefully)

How many Benchmarks

- A few (2 or 3) well chosen benchmarks should capture the major performance differences for data access and CPU related performance aspects
- Additional benchmark are likely to simply add “noise” to the data

Key Features

- Only a small number of benchmarks
- Synthetic application benchmarks
- Well defined data access structures
- Pencil and paper benchmark descriptions
- Reference implementations
- Scalable problem sizes
- Performance attributes independent of size
- Reasonable Run times

“More Specific Step”

- Research on characterization and development of benchmark need to go parallel (iterative)
- Basic strategy is to get a “prototype” of the first benchmark out in “6 month” and start collecting “feedback”
- To be of value this benchmark has to represent a class of “tough” problems

“More Specific Step”

- “To get the right decisions made in non-user space:”
- “That’s a hard problem”
- Repository
- Linked to TOP500
- Get measurements

“Questions”

- What are the most critical and relevant performance factors of our (scientific-HPC) applications?
- What benchmarks are there?
- What is a good class of algorithms for the first take?
- What infrastructure do we need?

Summary

- To evaluate different HPC architectures for different application domains we need a frame of reference in performance space
- A better understanding of performance determining factors of algorithms would greatly help the design of such benchmarks
- Establishing a stable performance reference would benefit the HPC community greatly